

Pascal NEWSLETTER

NUMBER 5

OREGON SOFTWARE

NOVEMBER 1982

SourceTools ready for field test

SourceTools is designed to manage program development in situations where a wide range of products requires numerous source files, where programmers are engaged in cross-system software development, or where several programmers may be working on the same program modules.

Field test will be conducted on the RSX and VMS (compatibility mode) operating systems. Our field-test price for SourceTools will be \$2,700 — at least 25% lower than the end-user price. Make-up of the end-user product, and its release date, depends upon field-test results.

We developed SourceTools originally for our own internal use, to manage cross-development projects and update multi-version software, such as our Pascal-2 compilers and cross-compilers for various machines and operating systems. Extensive in-house use has convinced us that these tools can benefit other software shops.

Features

Added Control. When more than one person must obtain copies of a file or alter a program, SourceTools prevents independent authors from making simultaneous and conflicting changes.

A Useful History of Changes. For each change made to a source file under its control, SourceTools records who had it, what changes were made, and why.

Economy of Space. SourceTools stores only the original version of a file and the differences between the source and each successive version, instead of storing each version as a whole copy.

Economy of Time. During cross-development of software, sources are simultaneously maintained on both the host and the target systems. You can reduce transfer time by shipping only the differences between systems.

Efficiency and Accuracy In Re-Creating Software. One SourceTools program mechanizes the re-creation of software from its component modules and codifies the process of updating files. This assures developers that modules are never forgotten, that out-of-date modules are never accidentally incorporated, and that commands are never executed unless they are necessary. Executing a minimum number of commands is more efficient than using command files.

Contents

The SourceTools package contains three groups of programs

that work together as a package. The first group, Source Control (SourceCon), consists of four programs for controlling the creation and modification of source files. The individual programs provide these services:

- **NewSrc** creates a new module from a starting file, establishing control information and a "checksum" to ensure validity of updates or changes.
- **GetSrc** provides a user with a version of the source file that can be used or modified. When a source file is checked out for modification, GetSrc denies subsequent requests to modify the source, providing copies only, until the file is returned (see UpdSrc). By default, GetSrc retrieves the most current version, but any earlier version can be specified.
- **UpdSrc** re-installs as the current source an edited version of a copy that was checked out with GetSrc. UpdSrc records controlling information about the update (see PrtSrc), and automatically determines the differences between the current version and the file that was checked out originally.
- **PrtSrc** prints the stored information about the differences between any two versions of a module, including the name of the author, the time of the change, the author's written description of the changes, and the sequence of editing commands needed for conversion to a later version.

Using SourceCon programs, you can access all versions, and you can trace the history of all changes.

Continued on Page 2

In this issue...

SourceTools Ready	Page 1
Field test policy	Page 2
OPUS Communique	Page 2
Information exchange	Page 3
Letters	Page 3
Pascal programs as RT-11 System Jobs	Page 4
Reserving room for RT-11's USR	Page 5
Reducing a program's task image size	Page 9
Literal strings in structured constants	Page 8
The Log: Pascal-1 and Pascal-2	Page 8
Bug contest begins	Page 9
Customer survey	Page 11

1894

15th Nov 1894

Dear Sir

I have the pleasure to inform you that

the same has been forwarded to you

and I am sure you will find it of interest

I am, Sir, very respectfully,

Yours faithfully,

Wm. H. Smith

Secretary

to the

Committee

of the

Association

of the

Professors

SourceTools Continued from Page 1

The second group contains the two programs, TextCom and SEdit. TextCom can generate a script file for parallel source maintenance on different processors. Using the same efficient algorithm used in the SourceCon programs, TextCom compares two files and lists the differences between them. A command-line switch creates an editor script as output for use by a "stream editor" program called SEdit. As a file is read in, SEdit applies the editing commands contained in the script file to a designated file. In the cross-development environment, these two programs work together to minimize host-to-target transfer time. When transferring code from the host development system to the target system, you only ship a short editor script file between systems instead of transferring complete, often lengthy, source files.

TextCom may, of course, be used for any operation that involves text comparison; its use is not restricted to cross-development or generation of editor scripts. TextCom can make better comparisons between files with numerous differences than Digital Equipment's comparison program, DIFFERENCES.

The third component is a single program called Make, which automatically keeps files up to date as components are changed. The user supplies a description file containing the names of files that depend on others and the commands for rebuilding any given file. Make examines the dates of all related files, redoing any out-of-date file, according to the directions in the description file. The Make program executes the fewest number of commands needed to update the modules used in a given file.

Persons interested in becoming SourceTools field-test users should contact Pat Rau at Oregon Software for licensing details.

Field test policy

Oregon Software will be offering its new products to field-test sites for at least a 25% discount off the anticipated end-user price. In exchange for that, we are looking for users who have a thorough knowledge of the hardware/software system on which these programs are implemented and who can evaluate the performance of the new software and suggest ways it may be improved.

Field-test users trade off the hassles of testing software against the benefits of having the product early, getting it at a reduced price, and helping to determine the nature of the final product.

Test-site programmers must be able to cope with "untested" software, to reduce problems to simple examples of one page, and to resolve problems over the phone. Field-test sites must be willing to use preliminary documentation.

Field-test sites will receive the end-user release automatically. The granting of a field-test license, however, does not guarantee that Oregon Software will release an end-user product.

OPUS communiqué

Oregon Pascal Users Society

Starting with this issue of the newsletter, the newly formed Oregon Pascal Users Society (OPUS) will submit a column dedicated to the sharing of information between Oregon Software and its customers. OPUS is not affiliated with Oregon Software, though Oregon Software is encouraging the group's formation. Membership is free, so join now! Here's the address:

Oregon Pascal Users Society (OPUS)
Bruce Williams
c/o EOCOM
15771 Redhill Ave.
Tustin, California 92680
(714) 730-5051, ext. 302

The need for a society for Oregon Software Pascal users has been evident for a few years. OPUS is an independent organization whose goals are:

1. To share experience in methodology of software engineering, software quality assurance, and/or details of implementation using Oregon Software's Pascal compilers as well as other languages that interface to the Oregon Software Pascal compilers;
2. To disseminate this information through available, expedient and economical channels;
3. Thereby providing a communications channel through which Oregon Software, members of OPUS, and members of external organizations can communicate approaches, techniques, problems, tricks and standards that may speed project development involving Oregon Software's Pascal compilers.

OPUS is being formed by a company in southern California and membership requests have come from as far away as Tasmania. What does it take to become a member? A postcard, letter or phone call.

So far, OPUS has helped users of Oregon Pascal (versions 1.1, 1.2, 2.0 for RT, RSX, RSTS/E) to achieve things as simple as character I/O using RT-11 V2.0 and as complex as AST and event-flag-controlled routines through FORTRAN-called procedures under RSX. If you don't have the NIH ("not invented here") problem, if you think you have something to share or a need someone may have already met, then call or drop a note to OPUS. That will automatically make you a member.

Future communiqués will describe what OPUS members have been working on and report their successes/failures in reaching their goals.

Information exchange

If you need information on technical applications involving Pascal, or if you have an application that might interest other users, send us a brief description for inclusion in the Information Exchange. Your description should follow the format of the items below. Interested parties can contact one another directly.

MEASURE, a utility program for use with Pascal-2 under the RT-11 operating system, measures execution time of procedures, functions, and statements, producing a profile of the program in terms of the time spent in execution. E. J. Sauter, Processing Concepts Limited, 2909 North Sheridan Road, Chicago IL 60657, (312) 883-1444.

Letters

To the Editor:

Occasionally one reads something which compels one to write a letter of clarification. Such an item is your statement (on page one of the August 1982 issue) that you are releasing the MC68000 OEM cross-compiler now because that processor "does not yet have a good development environment." Your readers should be aware that this statement is factually incorrect.

Despite its potential difficulties as an end-user environment, UNIX is widely acknowledged as a superior development environment. Unisoft of Berkeley has implemented a full native-mode UNIX on the MC68000, with the standard tools from Bell Labs' V7 UNIX as well as most of the University of Berkeley enhancements.

The Unisoft UNIX is available now with the Dual Systems Control Corporation (also in Berkeley) System 83, an IEEE-6 standard (S-100) MC68000 system. Unisoft UNIX is also apparently available for the Sun Workstation and the WICAT, both MC68000-based small systems, and is available to OEMs for inclusion in OEM packages. There are other UNIX versions for the 68000 as well, but I am most familiar with this particular one.

There is at least one "good development environment" for the MC68000.

Ian F. Darwin
Software Supervisor
Communications and Small Systems
University of Toronto Computing Systems

Pascal Programmers sought by Fortune 500 company with a central data center and large on-line systems. The company is dedicated to using Pascal as a second language; assembler is now the primary language. Write: Information Industries, Inc., 1500 Charter Bank Center, Kansas City MO 64108, Attention: Larry McWilliams.

Software training specialist needed to conduct training classes for customers and employees. Subjects: programming in Pascal language, writing process-control programs, use of RT-11 operating system and utilities, supporting software for ESI laser-processing systems. Contact: Harve Howard, Electro Scientific Industries, Inc., 13900 NW Science Park Drive, Portland OR 97229, (503) 641-4141.

Editor's Reply:

In general, we agree with you that a genuine UNIX system would be a good development environment for the MC68000. The problem is finding one.

Eighteen months ago, when we began our MC68000 project, our statement was definitely true: no good 68000-based UNIX system existed. So we used the mature RSX-11 system to develop the cross-compiler. That remains a fine choice for PDP-based users.

Today, a bewildering array of MC68000-based "UNIX" systems is being promoted. These are similar to UNIX, or based on it, but not identical. Worse, many of them aren't really available yet. We are still monitoring the situation, looking for a UNIX system to use ourselves.

We are aware of Unisoft's UNIX. We did not use that system because it is available for a limited set of configurations, which does not include the EXORMacs. Also, running it requires a great deal of disk space. The system is good, as you say, but of limited availability.

We intend to support a UNIX system soon, but not until we see which of the many contenders becomes the established MC68000 development system.

Editor's Note:

We want to keep our mailing list for the newsletter current. So, please, when you move, let us know where to send your newsletter or ask us to cancel your subscription. To request additional copies or give us the name of a new recipient, contact the editor. Thanks.

Pascal programs as RT-11 system jobs

Mr. Sauter is with Processing Concepts Ltd., 2909 North Sheridan Road, Chicago, Illinois, where he is engaged in systems development using Pascal. His current project is a Relational Data Base Management System (RDBMS) using Pascal as the host language.

Although RT-11 Version 4 is nicknamed "Mini-Tasker", it has only one SYSGEN option, **QUEUE**, to support system jobs, and Digital Equipment Corporation does not recommend the use of other available job slots. For those who are tempted, as we are, to create system jobs, the XM monitor has two important features: virtual overlays and virtual jobs. Assuming you are familiar with the techniques for creating overlaid programs (both virtual and disk-based) explained in the *Pascal-2 Programmer's Guide*, Version 2.0 for RT-11, this article explains how we minimized the low-memory static window size to allow several Pascal programs to be run as system jobs.

First, remember that when a Pascal-2 program is first executed, the initialization code performs at least these steps:

1. Test for proper compiler version number;
2. Allocation of stack and heap, using the monitor's **.SETTOP** request;
3. Initialization of file channel areas.

Under all monitors, **.SETTOP** specifies a new program high address from the monitor, and Pascal-2 uses this new space for the stack and the heap. The stack is used to store local variables and return addresses, and to pass parameters to procedures and functions. The stack starts at the address specified in the **.SETTOP** instruction and expands downward. The heap begins just after the program image and grows upward toward the stack. The heap is allocated dynamically with Pascal's predefined procedure **new**. Memory is returned to the heap when files are closed or when the predefined procedure **dispose** is used to deallocate variables allocated via **new**.

In the XM monitor, **.SETTOP** operates differently than it does in the SJ and FB monitors, in that enabling the special **.SETTOP** requires two actions: linking with the **/XM** switch and making the job virtual. Thus, **.SETTOP** allocates memory (hence the stack and the heap) from extended memory. Pascal-2 allocates this space with the instruction

.SETTOP #-2

With XM executing the virtual job, the Pascal program has 32K words of address space.

With the stack and heap in extended memory, it would be useful to put the whole Pascal program there also. All virtual programs require a static region that must reside in low memory, to hold the root and low memory overlays. However, in order to minimize the amount of low memory required, overlays aren't used and the root is made as small as possible. Instead, create a module with a new transfer address that will reside in low memory but only consists of a jump to the Pascal-2 entry point. This assembly-code module consists of:

```
.GLOBL  $START
XMST:   JMP    $START
        .END
```

To link the program with the module, use the following instructions:

```
.LINK/XM/EXE:PROG/PROMPT/C
*/TRANSFER XMST.VIRJOB.PASCAL
*PROG/V:1//
Transfer address? XMST
```

This method puts much of the code into extended memory. Due to the way the Linker loads library code, the root still contains the initialization code and most of the other library support routines and its size remains 2600 words. For this technique to be successful, the reference to the library routines must be forced into the overlay region. Again, you use the assembly code routine:

```
.GLOBL  XMST
XMST1:  JMP    XMST
        .END
```

and link the program with these commands:

```
.LINK/XM/EXE:PROG/PROMPT/C
*/TRANSFER XMST1.VIRJOB.PASCAL
*XMST.PROG/V:1//
Transfer address? XMST
```

This puts all but the overlay handler with its table and the **JMP XMST** instruction into virtual overlay region 1. The size of the root has been reduced to less than 400 words, as you can see using the **SHOW JOBS** command.

Creating Pascal system jobs in this manner carries a couple of restrictions:

- The addition of any more segments and/or regions will put Pascal-2 initialization code and I/O support back into the root. The program then needs a minimum of 2400 words of low memory and must be linked differently.
- The allocation of 32K words of memory by the Pascal-2 support library **.SETTOP #-2** instruction restricts the number of jobs you can run. For example, on a 192K-byte system, memory can only hold three jobs (with the queue running) before there is insufficient memory.

- When the virtual high limit of these programs lies within 28K to 32K words of memory (page 7), the job fails during initialization at the call to `.SETTOP`. Although such cases are rare, these failures are caused by the manner in which `.SETTOP` allocates memory above the user program upon initialization.

Reserving room for RT-11's USR

Mr. Siemens is with Omnex Coporation, 801 East Charleston Rd., Palo Alto, CA. Omnex uses Pascal to develop system utilities and end-user applications. One such application is a local system network providing a disk server to a number of remote RT-11 systems.

This article describes NEWSTART, a small initialization routine that reserves enough space for the User Service Routine (USR) to remain resident in an RT-11 system with the single job (SJ) monitor. By eliminating the USR swapping that occurs when a file `reset` or `rewrite` is performed, this routine speeds up the file opening process.

Under RT-11, programs compiled with either Pascal-1 or Pascal-2 are normally linked with a starting address at global symbol `$START`. At this address, the initialization code allocates the stack to be used by the program. The default stack is placed at the top of memory (a `.SETTOP` for all of memory is performed) unless the user has specified a stack address with the linker options `/STACK` (CCL syntax) or `/M` (CSI syntax). Actually, these options simply load a value that indicates the initial setting of the stack pointer into location 42 of the program. If the initialization code at `$START` finds that location 42 is greater than the high

Although DEC insists that RT is a single-user system, under certain circumstances we have found that it can almost look like a multi-user system. In this case, we have been able to run data entry applications and large compute-bound jobs that call `.TWAIT`, allowing background programs time to run also. We intend to use this approach for multiterminal applications in the near future.

program limit, then it will use the value in location 42. Otherwise, it will ask for all of memory.

The NEWSTART routine first tests to see which operating system is being used. If it is TSX-Plus or a non-privileged virtual job under the XM monitor, then control is immediately transferred to the regular start location, `$START`. If it is RT-11 SJ and `SET USR SWAP` is in effect (allowing the USR to swap), then location 42 is set with the normal USR load address. The routine at `$START` will then allocate the stack below the USR.

The initialization routine is simply linked with the program and a specific transfer address to global symbol `NEWSTA` specified. For example, assume the routine exists in the object file `NEWSTR.OBJ`, which contains global symbol `NEWSTA` where the program should start. The following commands link a program with `NEWSTR`:

```
LINK/TRANSFER PROG.NEWSTR.SY:PASCAL
Transfer symbol? NEWSTA
```

As a demonstration, the following short Pascal-2 program opens an output file 10 times and measures the elapsed time required. The table below the program shows the measured execution times for this program with and without `NEWSTR` on three different operating systems.

```
program Stest;
var
  I: integer;
  F: text;
  T1, T2: real;
begin
  T1 := time;
  For I := 1 to 10 do
    begin
      rewrite(F, 't.tmp'); requires use of USR
      close(F);
      end;
  T2 := time;
  writeln('Elapsed time', 3600.0(T2 - T1): 6: 2, ' seconds');
end.
```

Measured Execution Times in Seconds

Operating System	with NEWSTR	without NEWSTR
TSX-Plus	1.39	1.39
RT-11SJ	1.00	8.85
RT-11FB background	1.00	9.63

RT-11 USR Continued from Page 5

The source code for the new initialization routine follows. Note that it does not occupy any extra space in the save image file because it resides in psect RTSDAT (an overlaid psect). After NEWSTA has run, the code at \$START loads data over it.

```
.TITLE  NewStart for pascal programs
.Enable 1c
```

```
-----
; Module:      NewStart
;
; File:        NEWSTR.MAC (Source)
;              NEWSTR.OBJ (Object)
;
; Last Chng:   18-Aug-82   9:16 AM
;
; Purpose:     This module starts Pascal-1 and Pascal-2 programs
;              in such a way as to leave room in high memory for
;              the USR. This will keep USR swapping from occurring
;              when files are opened without forcing the user to do
;              an explicit SET USR NOSWAP.
;
;-----
```

```
; When linked such that the symbol NEWSTA is the transfer address,
; this routine makes sure that sufficient space is left in high memory
; for the USR to remain resident. This space is reserved only
; if the USR is not already available.
```

```
.PSECT RTSDAT, RW, I, GBL, REL, OVR
```

```
; This psect contains data necessary for the RT-11 interface.
; It is an overlaid psect, and since it contains no valid data until
; initialized, we can use the space for this one time only
; initialization code.
```

```
.GLOBL $START, RtArea
.MCall .GVal, .Serr, .Herr
```

FrstStk	=	42	;contains initial top of stack
Jsw	=	44	;job status word
VirJob	=	20000	;virtual job bit in jsw
UsrLdAdd	=	266	;offset for USR load address
Config	=	300	;offset for configuration word
UsrLkd	=	1000	;USR locked indicator bit

1-27-44 (12-1-71)

ALL INFORMATION CONTAINED
HEREIN IS UNCLASSIFIED
DATE 1-27-44 BY 12-1-71

CONFIDENTIAL

ALL INFORMATION CONTAINED
HEREIN IS UNCLASSIFIED
DATE 1-27-44 BY 12-1-71

ALL INFORMATION CONTAINED
HEREIN IS UNCLASSIFIED
DATE 1-27-44 BY 12-1-71

CONFIDENTIAL

ALL INFORMATION CONTAINED
HEREIN IS UNCLASSIFIED
DATE 1-27-44 BY 12-1-71

CONFIDENTIAL

ALL INFORMATION CONTAINED
HEREIN IS UNCLASSIFIED
DATE 1-27-44 BY 12-1-71


```

NewStart::
    .Serr                                ;soft errors desired
    Mov    #TsxGln,RO
    Emt    375                            ;try to get TSX-Plus line number
    Bcc    1$                            ;it worked, don't need to reserve
                                           ;any room for USR with TSX

    Bit    #VirJob,0#Jsw                 ;check for virtual job under XM
    Bne    1$                            ;it is, USR is available under XM

    Cmp    #1000,0#FrstStk               ;check for initial stack = 1000
    Bne    1$                            ;branch if it is
    .GVal  #RtArea,#Config
    Bit    #UsrLkd,RO                   ;check for USR locked
    Bne    1$                            ;branch if it is
    .GVal  #RtArea,#UsrLdAdd             ;get USR load address
    Dec    RO
    Dec    RO
    Mov    RO,0#FrstStk                 ;set initial stack
1$:    .Herr                             ;enable hard errors again
    Jump   $START                       ;go to normal run-time initialization

TsxGln: .Byte 0,110                     ;code for get line number

    .End

```

Reducing a Pascal-2 program's task image size

Users who are running out of disk space (who isn't?) can use an obscure Task Builder option to reduce the size of their Pascal-2 task image files. The room saved depends on the size of the global data area in the program. Consider the following short program:

```

program Test;

var I: integer;
    A: array [1..20,000] of integer;

begin
    for I:=1 to 20,000 do
        A[I]:=I;
    end.

```

After this program is compiled, it can be linked in the usual manner, using the command:

```
>TKB TEST/FP/CP,TEST=TEST,LB:[1,1]PASLIB/LB
```

The size of the task image is 92 blocks. The image is so large because the array A allocates 20,000 words of storage in the global data area, represented by the program section called GLOBAL. However, Pascal does not initialize this data area. This means that the data area need not be present in the task image. To get rid of the allocation for the psect GLOBAL, create the following overlay description file (TEST.ODL).

```

.NAME PASDAT,NODSK
.PSECT GLOBAL,D,GBL,OVR
.ROOT TEST-LB:[1,1]PASLIB/LB,PASDAT-GLOBAL
.END

```

The .NAME directive gives a name to a segment (called PASDAT here) and assigns the NODSK attribute to that segment. This means that the Task Builder will not allocate disk space for this segment. The .PSECT directive defines the program section GLOBAL in which Pascal will allocate all of the global level variables. The psect is named here so we can force it into the PASDAT segment. This is done by creating a co-tree in the .ROOT directive. The effect is that no disk space will be allocated for the co-tree called PASDAT. This causes no problems for the program, because all of the data in the section GLOBAL is initialized at run-time.

Use the /MP switch to tell the Task Builder to use the overlay description in the file TEST.ODL and build the task as shown below:

```
>TKB TEST/FP/CP,TEST=TEST/MP
```

Now the size of the task image file is 14 blocks. This is a considerable savings in disk space! This technique does not save any memory. In fact, the task grew by about 100 words by the addition of the co-tree. This trick works because the Pascal data area does not need to be read in from the disk to be initialized. The task loader allocates the space automatically when the task is activated.

Using literal strings in structured constants

As some users have noticed, Pascal-2's output files contain duplicate entries of literal strings that are contained within structured constants. This is due to the compiler's multi-pass design. On the first pass, all strings are placed in the output file. During the second pass the structured constant is processed and the entire constant value is placed in the output file, including a copy of any literal strings.

Example 1: Regular Use of Structured Constants

Input code in program:

```
type
  Rec = record
    Txt: packed array [1..12] of char;
    Len: 0..12;
  end;
{$nostandard}
const
  Duplication = Rec('duplicate ', 9);
{$standard}
begin
end.
```

Output code in the CONST psect:

```
.psect consts,d,con,lcl
$const:
.word 72544 ;'ud'  --literal string
.word 66160 ;'lp'
.word 61551 ;'ci'
.word 72141 ;'ta'
.word 20145 ;' e'
.word 20040 ;
.word 72544 ;txt = 'ud' --structured const
.word 66160 ; 'lp'
.word 61551 ; 'ci'
.word 72141 ; 'ta'
.word 20145 ; ' e'
.word 20040 ;
.word 11 ;len = 9
```

Users who require absolutely minimal-size programs, e.g. for ROM applications, may use this simple work-around. Instead of defining the strings as strings, drop down one level of structuring and define the individual characters of the array as separate elements. This method requires longer input code but generates only one copy of literal strings, producing shorter executable code in the CONSTS psect, as the following two examples show.

Example 2: The Work-around

Input code in program:

```
type
  Rec = record
    Txt: packed array [1..12] of char;
    Len: 0..12;
  end;
{$nostandard}
const
  NOduplication = Rec(('n', 'o', 'd', 'u',
                        'p', 'l', 'i', 'c',
                        'a', 't', 'e', ' '),
                      11);
{Note the additional parentheses needed
to indicate a lower level of structuring.}
begin
end.
```

Output code in the CONSTS psect:

```
.psect consts,d,con,lcl
$const:
.word 67556 ;'on' --structured const
.word 72544 ;'ud'
.word 66160 ;'lp'
.word 61551 ;'ci'
.word 72141 ;'ta'
.word 20145 ;' e'
.word 13 ;len = 11
```

The Log: Pascal-1 and Pascal-2

The Log is empty for this issue. We are still releasing version K of both Pascal-1 and Pascal-2 as our current product. If you have any problems with version J or before, and you are still in support, be sure to request your free update. We have not forgotten those of you who have submitted Trouble Reports for Version K. At this very moment, we are busily fixing bugs for Pascal-1 Version 1.3 and Pascal-2 Version 2.1.

In addition, we are adding a number of new features to Version 2.1, such as an improved interface to the Pascal support library, improved error reporting, and conformant arrays.

When the Support Group at Oregon Software gets several calls about a particular problem, we do one of two things. We fix bugs, announcing the fixes in the Log, and we publish articles in the newsletter, to clarify the use of Pascal-1 or Pascal-2 under the various operating systems, or to offer work-arounds for specific users' problems. Your letters help us to be aware of current problems facing users, to provide specialized information on the operating systems we support, and to develop the support libraries for our products. So, if you don't find information you need in the newsletter, write us.

Editor's note: The yearly support fee for Pascal-2 is \$900; Pascal-1 support is \$600 annually.

Bug contest begins

Why let program bugs bring you nothing but grief? Enter Oregon Software's Bug Contest and win a prize for the most interesting bug you've created while writing programs using our software.

What do I enter? You can enter the contest in any or all of three categories: most interesting bug, most interesting application program, most interesting prize.

We all know what a bug is. A mistake or oversight in program logic can be subtle and conditional, yielding unanticipated or undesired results. Such bugs are intellectually interesting as puzzles. What makes one of them more interesting than another? The bug really captivates us when it causes truly spectacular results, like turning all the traffic lights in the city red for an hour.

An interesting application could be an innovative use of Pascal, some heretofore unrecognized characteristic of the language finding its place in an area of the real world where Pascal is not usually found. Or, a program could be interesting because of its unusual results, the job it accomplishes. Or, the code of the program could be interesting in its own right, because the algorithm or its coding is so clever and elegant. We'll accept all three types.

Finally, the prize suggestions will be interesting because of the thought, not the expense. The best birthday gift is "right" because it fits the person better than anything that person would have asked for. We look for suggestions that are exquisite, appropriate, and attainable. Because the winner of each category gets one of the top three prizes, we hope your suggestions are not so unique that we can't obtain them.

How do I enter? You'll notice that we've set up the entry blank in three parts: the top two for events and applications, and the bottom for prize suggestions. You may enter all three, and you may submit as many entries as you like. Make as many photocopies of the entry form as you need.

In the description of your bug and/or application, please include these bits of information:

- We'd like to know the environment of your specimen. What conditions caused the bug to break out of its cocoon and take flight?
- Where and when did you develop the application program? What problem did the program solve?
- Were there any unusual aspects of your hunt for the bug or your development of the application program?

Other bug catchers will be curious about the handling of such pests, so please tell us what you did to squash your bug, and what you think others could do to avoid such occurrences.

And if I win? For those of you who don't think the fun of entering is its own reward, we've that third category for suggested incentives. We plan to give the three most interesting prizes to each winner. The keeper of the most interesting bug will be awarded the most interesting prize suggestion. The second place prize suggestion will go to the submitter of the winning application program. The person who suggests that most interesting prize gets the third most interesting prize (after all, you don't want to know what you're getting, do you?).

When is my entry due? All entrants and the winners will be announced in Newsletter #7 (June 1983). So, we need to have your entry by March 1, 1983. We already have several entries, so don't delay. Send yours now!

Pascal NEWSLETTER

OREGON SOFTWARE
2340 SW Canyon Road
Portland, Oregon 97201

Collins Hemingway, editor

David Spencer and Michael Kuhn, staff writers

The Pascal Newsletter is published quarterly by Oregon Software, Inc., 2340 SW Canyon Rd., Portland, OR 97201; (503) 226-7760. Each customer of Oregon Software receives one free subscription per site. Additional subscriptions are available upon written request.

The Pascal Newsletter accepts articles of interest to Pascal users: solutions to troublesome programming situations, new applications of Pascal, interesting variations on standard applications, etc. Submit articles on paper (typed and double-spaced), on floppy disk, or on magnetic tape, sent to the attention of the editor. Fee: \$100 per newsletter page. Payment upon publication.

Copyright © 1982 by Oregon Software, Inc.
ALL RIGHTS RESERVED.

RSTS, RSX, RT-11, PDP-11, VAX/VMS, and IAS are trademarks of Digital Equipment Corp. MC68000 is a trademark of Motorola, Inc. Pascal-1, Pascal-2, and Pascal Newsletter are trademarks of Oregon Software.

Printed in USA

ENTRY BLANK

Name _____

Site # _____

Company Name _____

Description of Entry:

BUG



APPLICATION



PRIZE



CONSUMER SURVEY

This survey of Oregon Software's customers and other interested parties will provide information for us to use in planning over the next year. We look forward to receiving your answers promptly. To encourage participation, we are offering a \$100 discount off any of our products or services to everyone who returns a completed survey. To receive the \$100 coupon, fill in your name and address on the final survey question. To use the coupon, simply include it with any order and subtract the \$100 from the amount you owe us. The coupon expires May 30, 1983.

After completing the questionnaire, please follow the mailing directions below.

Mailing Directions

Domestic: Tear off the questionnaire section and fold it so that the BUSINESS REPLY MAIL permit faces out. Secure the open edge with staple or tape, then mail.

International: Please enclose the questionnaire pages in an envelope and return them to your distributor. If you do not have a distributor, return the questionnaire directly to us. (Unfortunately, we cannot supply prepaid postage for overseas respondents.)

Thank you in advance for your response.



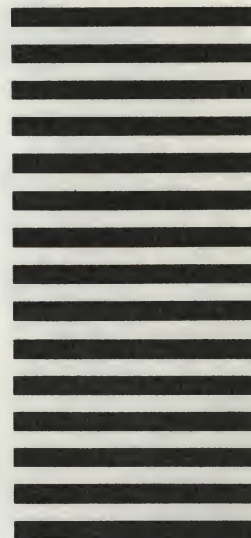
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. A407 PORTLAND, OR.

POSTAGE WILL BE PAID BY ADDRESSEE

OREGON SOFTWARE INC.
Attn: Marketing Survey
2340 SW Canyon Road
Portland OR 97201



1951 10/10/51

Dear Mr. [Name] -

I am writing to you regarding the [Topic] which you mentioned in your letter of [Date].

The [Topic] is a very important matter and I am sure that you will find the following information of interest.

I have been thinking about this matter for some time and I have decided to write to you about it.

I am sure that you will find the following information of interest.

I am sure that you will find the following information of interest.

I am sure that you will find the following information of interest.

I am sure that you will find the following information of interest.

I am sure that you will find the following information of interest.

I am sure that you will find the following information of interest.

I am sure that you will find the following information of interest.

I am sure that you will find the following information of interest.

1. What Oregon Software products does your organization currently have?

☐ Pascal-1
☐ Pascal-2
☐ Sort-1 or Sort-1-Plus
☐ None

2. On what processor/operating system is your Pascal compiler hosted?

☐ PDP-11/RSX ☐ PDP-11/RSTS
☐ VAX/VMS ☐ PDP-11/RT-11 or TSX
☐ PDP-11/RSX ☐ Does Not Apply

3. How important are the following Pascal compiler features to you?

	Unimportant			Important		
	1	2	3	4	5	
a. Code Size	1	2	3	4	5	N/A
b. Code Execution Speed	1	2	3	4	5	N/A
c. Compiler Execution Speed	1	2	3	4	5	N/A
d. Portability of Code	1	2	3	4	5	N/A
e. Adherence to Pascal Std.	1	2	3	4	5	N/A
f. Good Support	1	2	3	4	5	N/A
g. Ease of Debugging	1	2	3	4	5	N/A
h. Utilities Package	1	2	3	4	5	N/A
i. Competitive Price	1	2	3	4	5	N/A
j. Availability of Extensions	1	2	3	4	5	N/A
k. Other (please specify)_____	1	2	3	4	5	N/A

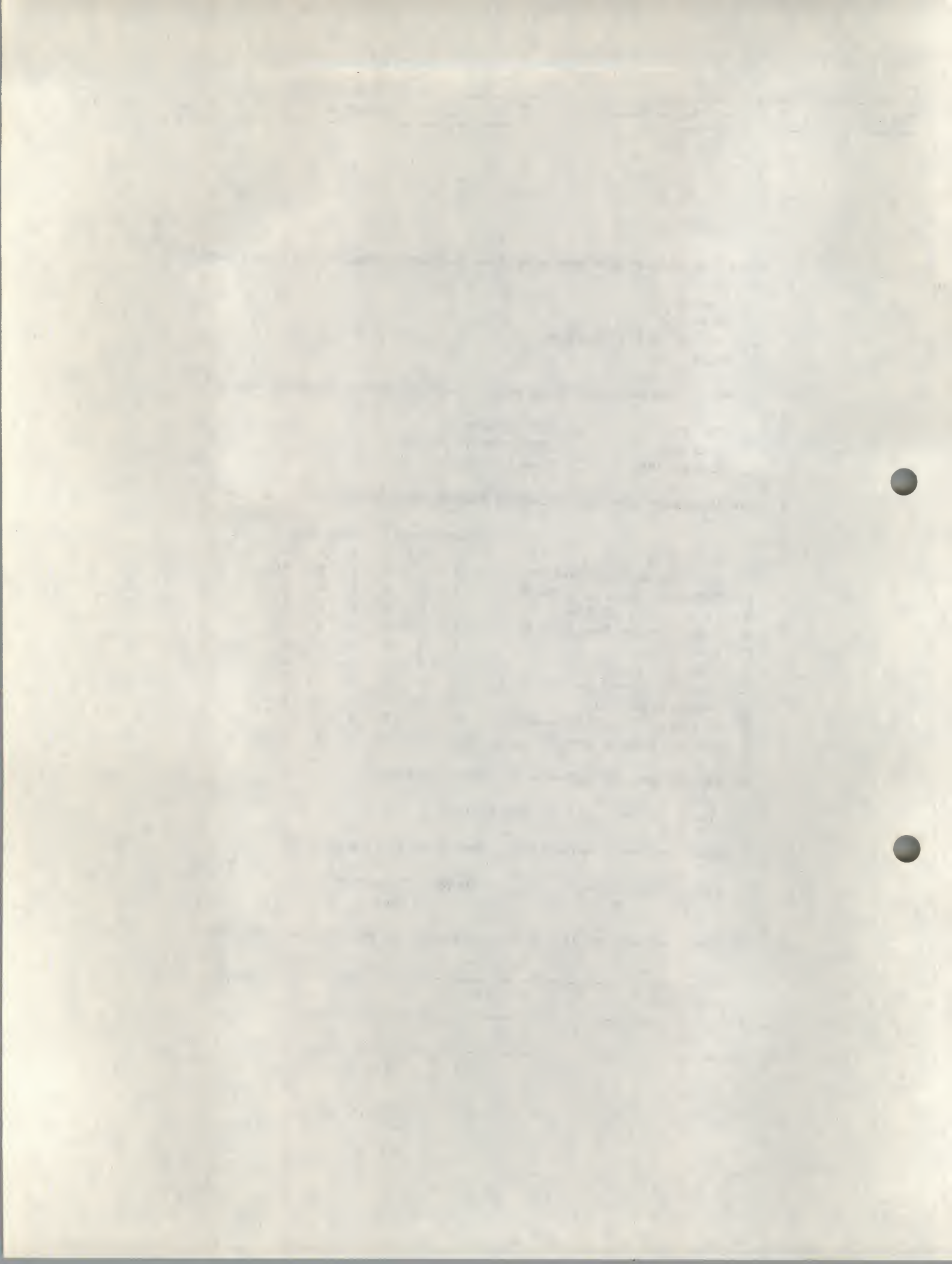
4. Do you use Oregon Software's support services?

☐ Yes ☐ No ☐ Does Not Apply

If yes, how satisfied have you been with the support?

Very Dissatisfied Very Satisfied
 1 2 3 4 5

If dissatisfied, or if you are no longer in support, explain why.



5. How would you compare the cost of completing a programming project using Oregon Software's Pascal versus another approach?

a. ___Far Below ___Below ___About the Same ___Above
 ___Far Above ___Unable to Determine

b. What would the other approach be? _____

6. Does Oregon Software's Pascal-2 have any serious limitations or drawbacks?

___Yes ___No ___Does Not Apply

Please explain (use extra sheet if necessary) _____

7. Please indicate the three most important reasons for choosing Oregon Software Pascal over another compiler or language.

1. _____ (First Priority)

2. _____ (Second Priority)

3. _____ (Third Priority)

8. What is the job title and function of the person most influential in deciding to acquire a compiler?

9. a. What is the basic activity at your facility?

___ Components/sub-assemblies
___ Computers/peripheral equipment
___ Communications systems/equipment
___ Aircraft/space/ground support equipment
___ Test/measurement/instrumentation/process control equipment
___ Other (please specify) _____

- b. What is the basic function performed at your facility?

___ Research and development
 ___ Hardware ___ Software
___ Manufacturing
___ Service and support
___ Other (please specify) _____

--- Commercial
 --- Government agency (non-military)
 --- Military
 --- Education
 --- Other (please specify) _____

a. ___ Under 5 ___ 11 to 20 ___ 30 to 50
 ___ 5 to 10 ___ 21 to 30 ___ More than 50

11. Please indicate how important the following languages are to you.

		Unimportant			Important		
		1	2	3	4	5	N/A
a.	FORTRAN	1	2	3	4	5	N/A
b.	BASIC	1	2	3	4	5	N/A
c.	Assembler	1	2	3	4	5	N/A
d.	Pascal	1	2	3	4	5	N/A
e.	COEOL	1	2	3	4	5	N/A
f.	C	1	2	3	4	5	N/A
g.	ADA	1	2	3	4	5	N/A
h.	PL/1	1	2	3	4	5	N/A
i.	Other (specify) _____	1	2	3	4	5	N/A

Did you have major influence on the decision to buy a compiler?

13. Where did you first hear about Oregon Software?

--- Friend or associate
 --- Trade show
 --- Professional association
 --- Distributor

--- Magazine article
 --- Magazine ad
 (which magazine? _____)
 --- In school
 --- Other (specify) _____

14. How important are the following sources for information on software?

	Unimportant			Important			
a. Vendor publications	1	2	3	4	5	N/A	
b. Distributors	1	2	3	4	5	N/A	
c. Sales representatives	1	2	3	4	5	N/A	
d. Magazines	1	2	3	4	5	N/A	
e. Trade shows	1	2	3	4	5	N/A	
f. Technical seminars	1	2	3	4	5	N/A	
g. Friends and associates	1	2	3	4	5	N/A	
h. Other (specify) _____	1	2	3	4	5	N/A	

15. What job-related publications do you read regularly?

16. To receive coupon:

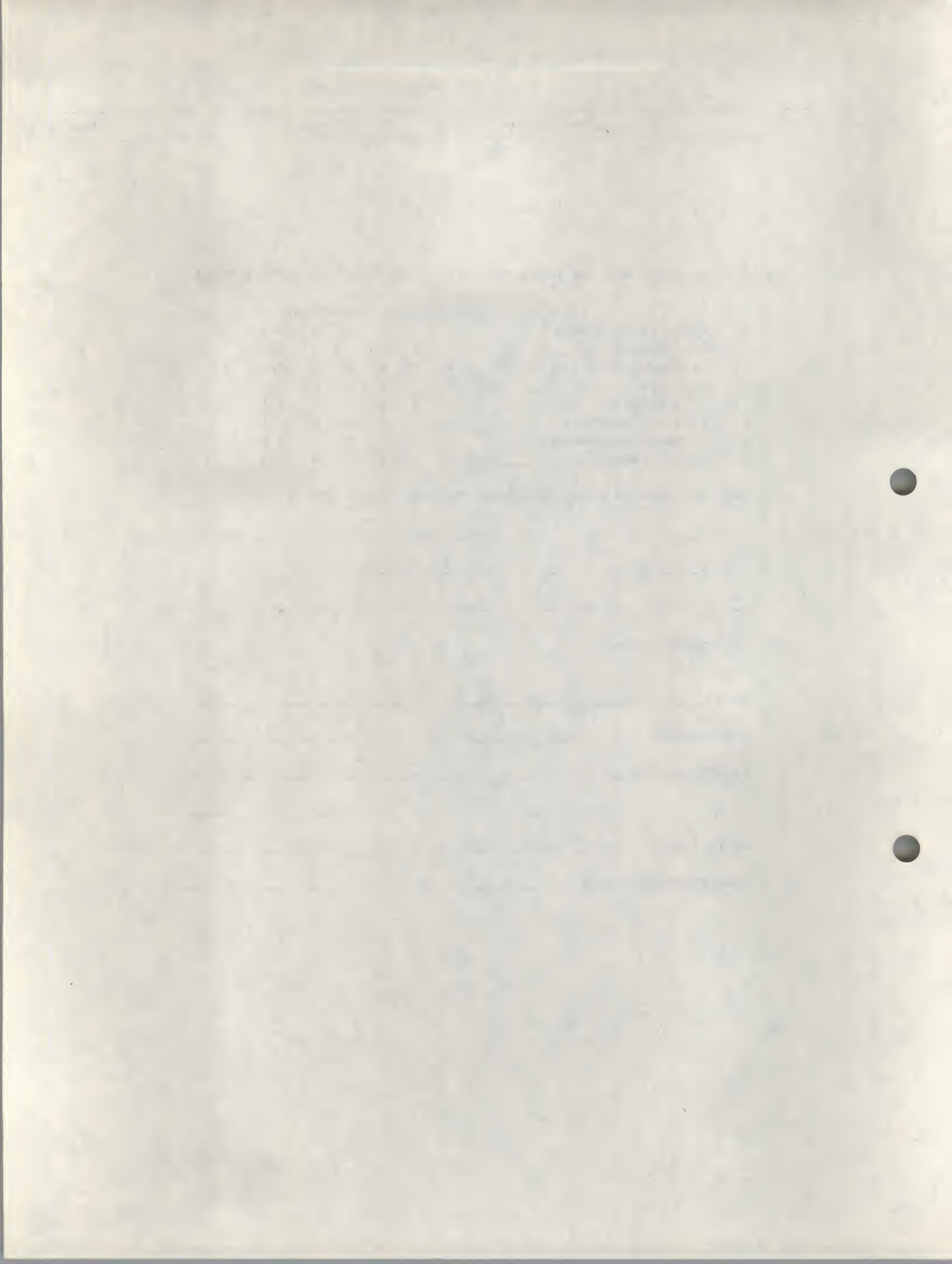
Name -----

Organization -----

Street Address -----

City, State -----

Country, Postal Code -----



Pascal
Newsletter
OREGON SOFTWARE

2340 SW Canyon Road
Portland, Oregon 97201

